IN THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Canceled)

2. (Currently amended) A method for verifying computer instructions in a computer that includes at least one processor that executes instructions stored in a memory, the memory being organized into separately addressable memory blocks, the method comprising:

identifying a next instruction of a series of instructions to be executed when executing [[a]] the series of instructions;

for the next instruction, and during the executing of the series of instructions, determining an identifying value for a memory block that contains the next instruction;

determining, during the executing of the series of instructions, whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition requires comparing the identifying value of the memory block with a set of reference values;

allowing execution of the next instruction by the processor when the identifying value satisfies the validation condition; and generating a protective response when the identifying value does not satisfy the validation condition~~; whereby the next instruction is verified dynamically before being executed~~.

3. (Previously presented) The method of claim 2, wherein the validation condition is that the identifying value of the memory block matches any reference value in the set of reference values.

4. (Previously presented) The method of claim 2, wherein the validation condition is that the identifying value of the memory block differs from each reference value in the set of reference values.

5. (Canceled)

6. (Currently amended) A method for verifying computer instructions in a computer that includes at least one processor that executes instructions stored in memory, the memory being organized into separately addressable memory blocks, the method comprising:

identifying a current instruction to be executed when executing a series of instructions, the current instruction being one of the series of instructions being executed identified for submission to the processor for execution and not yet executed at a time of the identifying;

for at least ~~one~~ the current current instruction ~~that has been identified for submission to the processor for execution~~, computing a hash value as a function of a ~~sub-set~~ subset of contents of a current memory block that contains the current instruction;

determining, during the executing of the series of instructions, whether the hash value satisfies a validation condition by comparing the hash value of the current memory block with a set of reference values;

if the hash value satisfies the validation condition, allowing continued execution of the ~~current instruction~~ the series of instructions; ~~by the processor;~~

if the hash value does not satisfy the validation condition, generating a protective response;

wherein the computing of the hash value comprises applying a mask to the current memory block, the mask being a data structure that designates at least one byte of the current memory block to be ignored in the computing of the hash value, the data structure designating less than an entire memory block so that the hash value is based on only part of the contents of the current memory block.

7. (Currently amended) The method of claim 6, further comprising:

identifying, ~~potentially non-constant contents~~ an indeterminate portion of the current memory block, the ~~non-constant contents~~ the indeterminate portion being ~~valid but changeable so that they do not indicate~~ non-indicative of validity of the current memory block as a whole; and

configuring the mask so that the mask designates at least the ~~non-constant contents~~ the indeterminate portion to be ignored when generating the hash value.

8. (Previously presented) The method of claim 2, further comprising:

for each of the separately addressable memory blocks, indicating in a structure whether the memory block is valid;

accessing the structure to determine whether the memory block is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the memory block is valid and directly allowing execution of the next instruction when the structure indicates that the memory block is valid.

9. (Previously presented) The method of claim 8, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of memory blocks is validated comprises setting one of the hardware attribute indicators, the one hardware attribute indicator corresponding to the memory block.

10. (Original) A method as in claim 9, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

11. (Previously presented) The method of claim 8, wherein the structure comprises a software data structure, and wherein the indicating in the structure whether the plurality of memory blocks is validated comprises making a corresponding entry in the software data structure.

12. (Previously presented) The method of claim 8, the determining of the identifying value for the memory block and the determining of whether the identifying value satisfies the validation condition are performed only when the structure does not indicate that the memory block is valid, the method further comprising:

modifying the structure so that the memory block is not indicated as being valid when the identifying value satisfies the validation condition.

13. (Previously presented) The method of claim 12, further comprising:

sensing modification of one of the memory blocks that the structure indicates is valid and, in response to the modification, setting its indication in the structure to indicate that the memory block is not valid.

14. (Previously presented) The method of claim 8, further comprising:

determining a branch history for the next instruction; and

checking whether the memory blocks in which instructions in the branch history are located are valid, the validation condition including the requirement that each checked memory block in the branch history is valid.

15. (Previously presented) The method of claim 2, wherein the determining of the identifying value and the determining as to whether the validation condition has been satisfied are performed only after a triggering event occurs.

16. (Previously presented) The method of claim 15, wherein the triggering event is writing of at least one new unit of code or data to any physical component within the computer.

17. (Previously presented) The method of claim 15, in which the triggering event is an attempted execution of any instruction located on any unverified memory block.

18. (Previously presented) The method of claim 15, wherein the triggering event is an attempted execution of any instruction located on any unverified memory block of newly installed software.

19. (Previously presented) The method of claim 15, further comprising triggering the verification of the computer instructions depending on an identity of a user of the computer, the user having caused the next instruction to be identified for execution.

20. (Canceled)

21. (Previously presented) The method of claim 15, further comprising triggering dynamic verification depending on a context in which the next instruction is submitted for execution, wherein the context is a level of security clearance associated with the computer, a user of the computer, or a program of which the next instruction is a part.

22. (Previously presented) The method of claim 2, wherein the identifying of the next instruction is performed for only a sample of the series of instructions.

23. (Currently amended) The method of claim 22, wherein the sample is a time-sampled ~~sub-set~~ subset of the series of instructions.

24. (Currently amended) The method of claim 22, wherein the sample is a sequentially sampled ~~sub-set~~ subset of the series of instructions.

25. (Currently amended) The method of claim 22, wherein the sample is a ~~sub-set~~ subset of the series of instructions sampled spatially, the sampling being over a range of memory block identifiers.

26. (Currently amended) The method of claim 2, wherein the protective response comprises termination of a software entity with which the current memory block is associated.

27. (Currently amended) The method of claim 2, wherein the protective response comprises suspension of execution of a software entity with which the current memory block is associated.

28. (Currently amended) The method of claim 2, wherein the protective response comprises a message posted to a user, system administrator, or other predetermined recipient.

29. (Currently amended) The method of claim 2, wherein:

the computer includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer; and

the protective response comprises a switching of an execution mode of the virtual machine from the direct execution mode to a binary translation mode.

30. (Currently amended) The method of claim 2, wherein:

the computer includes a virtual machine running on an underlying hardware platform via an intermediate software layer; and

the protective response includes checkpointing the state of the virtual machine.

31. (Currently amended) The method of claim 2, wherein the protective response is a first possible response, the method further comprising:

associating the first possible response with the memory block;

associating a second possible response with a different memory block;

upon detection of failure of the next instruction to satisfy the validation condition, identifying which one of the possible responses is associated with the memory block, and

generating the one possible response associated with the memory block in which the next instruction is located.

32. (Previously presented) The method of claim 2, further comprising:

associating reference values from the set of reference values with respective programs such that each association signifies that the reference value corresponds to a memory block storing instructions for one of the programs; and

tracking which of the respective programs is being executed and the association between the matching reference value and the corresponding one of the programs.

33. (Currently amended) The method of claim 2, wherein:

the computer includes a virtual machine (VM) running on an underlying hardware platform via an intermediate software layer [[is]] operable to switch the virtual machine between a direct execution mode and a binary translation mode; and

the series of instructions comprise VM-issued instructions issued in conjunction with binary translation of any of the VM-issued instructions, the VM-issued instructions thereby being verified.

34. (Canceled)

35. (Currently amended) A tangible medium embodying executable code ~~executable~~ for dynamically verifying a computer ~~instructions~~ instruction being executed by a processor of a computer, the executable code being a verification engine causing the computer to perform a method having operations of:

identifying a next instruction of a series of instructions to be executed when executing the series of instructions;

for the next instruction, and during the execution of the series of instructions, determining an identifying value for a memory block that contains the next instruction;

determining whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition requires comparing the identifying value of the memory block with a set of reference values;

allowing execution of the next instruction by the processor when the identifying value satisfies the validation condition; and generating a protective response when the identifying value does not satisfy the validation condition~~; whereby the next instruction is verified dynamically before being executed~~.

36. (Previously presented) The tangible medium of claim 35, wherein the validation condition is that the identifying value of the memory block matches any reference value in the set of reference values.

37. (Previously presented) The tangible medium of claim 35, wherein the validation condition is that the identifying value of the memory block differs from each reference value in the set of reference values.

38. (Previously presented) The tangible medium of claim 35, wherein: the identifying value is a hash value that is computed as a function of contents of the memory block.

39. (Currently amended) The tangible medium of claim 38, wherein some of the contents of the memory block are ignored when computing the hash value, the ignored portion being defined by a ~~sub-set~~ subset selection structure.

40. (Currently amended) The tangible medium of claim 39, wherein the ~~sub-set~~ subset selection structure is a mask.

41. (Previously presented) The tangible medium of claim 35, wherein the memory block is one of a plurality of separately addressable memory blocks and the method further comprises:

for each of the separately addressable memory blocks, indicating in a structure whether the memory block is valid;

accessing the structure to determine whether the memory block is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the memory block is valid and directly allowing execution of the next instruction when the structure indicates that the memory block is valid.

42. (Previously presented) The tangible medium of claim 41, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of memory blocks is valid comprises setting one of the hardware attribute indicators corresponding to the memory block.

43. (Previously presented) The tangible medium of claim 42, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

44. (Previously presented) The tangible medium of claim 35, wherein the structure comprises a software data structure, and wherein further comprising a software module comprising computer-executable instructions for selecting for verification only a sample of the ~~current~~ next instructions.

45. (Canceled)

46. (Previously presented) The tangible medium of claim 35, wherein:

the verification engine resides in an intermediate virtualization layer between a virtual machine and a hardware platform of the computer;

the next instruction is issued by the virtual machine; and

the verifying of the validity of the next instruction is performed while copying or translating in conjunction with binary translation of instructions for the virtual machine.

47. (Currently amended) The tangible medium of claim 46, wherein the protective response comprises switching the intermediate virtualization layer to a binary translation mode.

48. (New) The method of claim 2, wherein the identifying value is a hash value that is computed as a function of all contents of the memory block.

49. (New) The method of claim 2, wherein the identifying value is a hash value that is computed as a function of contents of the memory block such that some of the contents of the memory block are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.